Augmented Data Structures

• Why is it needed?
  - Because basic data structures not enough for all operations
  - storing extra information helps execute special operations more efficiently.

• Can any data structure be augmented?
  - Yes. Any data structure can be augmented.

• Can a data structure be augmented with any additional information?
  - Theoretically, yes.

• How to choose which additional information to store.
  - Only if we can maintain the additional information efficiently under all operations. That means, with additional information, we need to perform old and new operations efficiently maintain the additional information efficiently.
How to augment data structures

1. choose an underlying data structure
2. determine additional information to be maintained in the underlying data structure,
3. develop new operations,
4. verify that the additional information can be maintained for the modifying operations on the underlying data structure.
Interval Trees

• **Need**: *Dynamic* data structure to store time intervals
• **Application**: Maintain schedule for set of seminars
• **Operations**: Insert, Delete
• Every interval $j$ has: $\text{low}[j], \text{high}[j]$
• **Data Structure**:
  - Augment RB-Tree so that it can store intervals.
  - Ordering based on what key? low values? high values? $(\text{high}+\text{low})/2$ values? $(\text{high}-\text{low})$ values?
  - Note that insert and delete are still efficient.
• **New Operation**: Search (find any overlapping interval)
  - Problem with Search!
Augmented Information

- low, high, max
- \( \text{max}[x] = \) rightmost high value of all intervals in subtree rooted at \( x \)
- The value \( \text{max}[x] \) of each node can be written as:
  \[
  \text{max}[x] = \text{Max} \{ \text{high}[\text{int}[x]], \text{max}[\text{left}[x]], \text{max}[\text{right}[x]] \}
  \]
- Therefore it can be maintained efficiently under insertions and deletions
INTervals-Search

**INTERVAL-SEARCH** (T, j)
// finds an interval in tree T that overlaps interval j, 
// else return NIL.
1. \( x = \text{root}[T] \)
2. while \( x \neq \text{NIL} \) and j does not overlap \( \text{int}[x] \) do
3. \quad if \( \text{left}[x] \neq \text{NIL} \) and \( \max[\text{left}[x]] \geq \text{low}[j] \) then
4. \quad \quad \quad \quad \quad x = \text{left}[x]
5. \quad else \quad x = \text{right}[x]
6. \quad return x

**Time Complexity** \( O(\log n) \)
Greedy Algorithms

- Given a set of activities \((s_i, f_i)\), we want to schedule the maximum number of non-overlapping activities.
- **GREEDY-ACTIVITY-SELECTOR** \((s, f)\)
  1. \(n = \text{length}[s]\)
  2. \(S = \{a_1\}\)
  3. \(i = 1\)
  4. **for** \(m = 2\) **to** \(n\) **do**
  5. \(\text{if } s_m \text{ is not before } f_i \text{ then}\)
  6. \(S = S \cup \{a_m\}\)
  7. \(i = m\)
  8. **return** \(S\)
• [1,4], [3,5], [0,6], [5,7], [3,8], [5,9], [6,10], [8,11], [8,12], [2,13], [12,14] -- **Sorted by finish times**
• [1,4], [3,5], [0,6], [5,7], [3,8], [5,9], [6,10], [8,11], [8,12], [2,13], [12,14]
• [1,4], [3,5], [0,6], [5,7], [3,8], [5,9], [6,10], [8,11], [8,12], [2,13], [12,14]
• [1,4], [3,5], [0,6], [5,7], [3,8], [5,9], [6,10], [8,11], [8,12], [2,13], [12,14]
• [1,4], [3,5], [0,6], [5,7], [3,8], [5,9], [6,10], [8,11], [8,12], [2,13], [12,14]
• [1,4], [3,5], [0,6], [5,7], [3,8], [5,9], [6,10], [8,11], [8,12], [2,13], [12,14]
Why does it work?

• **THEOREM**
  Let $A$ be a set of activities and let $a_1$ be the activity with the earliest finish time. Then activity $a_1$ is in some maximum-sized subset of non-overlapping activities.

• **PROOF**
  Let $S'$ be a solution that does not contain $a_1$. Let $a'_1$ be the activity with the earliest finish time in $S'$. Then replacing $a'_1$ by $a_1$ gives a solution $S$ of the same size.

  Why are we allowed to replace? Why is it of the same size?